Test 1:

Enter first pair: A13
Enter second pair: B5
Enter third pair: A6

Output is A6 A13
Output is B5

Test 2:

Enter first pair: A1
Enter second pair: B9
Enter third pair: C6

Output is A1
Output is B9
Output is C6

Retesting:

All of the above tests plus the following:

Test 3:

Enter first pair: B11
Enter second pair: B9
Enter third pair: B6

Output is B6 B9 B11
Test Cases

Test 1:
   Enter final score: 7
   Pushups: 7

Test 2:
   Enter final score: 13
   Pushups: 30

Test 3:
   Enter final score: 37
   Pushups: 242

Retesting:

All of the above tests plus the following:

Test 4:
   Enter final score: 0
   Pushups: 0
Test 1:

Enter the number of points: 5
Enter x0: 1.1
Enter y0: 1.1
Enter x1: -.3
Enter y1: .2
Enter x2: 2.5
Enter y2: -.2
Enter x3: 1
Enter y3: -3.5
Enter x4: 1.5
Enter y4: .8
These points form a convex polygon.

Test 2:

Enter the number of points: 4
Enter x0: 2
Enter y0: -5.3
Enter x1: -6
Enter y1: 3.7
Enter x2: 1.2
Enter y2: 6.3
Enter x3: 2.7
Enter y3: -7
These points do NOT form a convex polygon.

Test 3:

Enter the number of points: 3
Enter x0: 1
Enter y0: 1
Enter x1: 3
Enter y1: 3
Enter x2: 2
Enter y2: 2
These points do NOT form a convex polygon.

Retest: Use all of the above tests, plus:

Test 4:

Enter the number of points: 4
Enter x0: 1
Enter y0: 0
Enter x1: 2.1
Enter y1: 1.2
Enter x2: 1.1
Enter y2: 1.4
Enter x3: 1.9
Enter y3: -0.2
These points form a convex polygon.
Input and output elements must be labeled as shown.

Test 1:

Enter dimension: 4  
Enter a41: 3  
Enter a31: 7  
Enter a21: -2  
Enter a11: 6  
Enter a12: 0  
Enter a13: 2  
Enter a14: 8  
Enter b1: 3  
Enter b2: -5  
Enter b3: 1  
Enter b4: 9  
c1 = 92  
c2 = -18  
c3 = 37  
c4 = 26

Test 2:

Enter dimension: 3  
Enter a31: 0  
Enter a21: 0  
Enter a11: 1  
Enter a12: 0  
Enter a13: 0  
Enter b1: 4  
Enter b2: 5  
Enter b3: 6  
c1 = 4  
c2 = 5  
c3 = 6

Retest: The above tests, plus:

Test 3:

Enter dimension: 4  
Enter a41: 9  
Enter a31: 8  
Enter a21: 7  
Enter a11: 6  
Enter a12: 5  
Enter a13: 4  
Enter a14: 3  
Enter b1: 2  
Enter b2: 3  
Enter b3: 4  
Enter b4: 5  
c1 = 58  
c2 = 72  
c3 = 86  
c4 = 100
5 Advanced – Patterns
Test Cases

Note that many patterns are acceptable. You will need to calculate the row and col sums for each row and col. None of those can be the same as the restricted value.

Test 1:
   Enter start value: 1
   Enter restricted value: 10
   Output: 7 8 9
           4 6 2
           3 1 5

Test 2:
   Enter start value: 10
   Enter restricted value: 42
   Output: 10 11 12
           15 18 13
           16 17 14

Retesting

All of the above plus

Test 3:
   Enter start value: 5
   Enter restricted value: 38
   Output: 5 6 7
           8 9 10
           11 12 13
Test 1:

Enter level number: 6
Enter position: 30
7/2

Test 2:

Enter level number: 20
Enter position: 300000
1279/936

Test 3:

Enter level number: 8
Enter position: 129
1/0

Retest: All of the above, plus:

Test 4:

Enter level number: 15
Enter position: 1000
38/159
1 Beginning – Sorting
Test Cases

Test Case 1:

Enter first integer: 13
Enter second integer: 5
Enter third integer: 6

Output is 5 6 13

Test Case 2:

Enter first integer: 12
Enter second integer: 12
Enter third integer: 15

Output is 12 12 15

Second Try:

Repeat the above test cases plus

Test Case 3:

Enter first integer: 4
Enter second integer: 400
Enter third integer: 40

Output is 4 40 400
2 Beginning - Willie's Pushups
Test Cases

Test 1:
Enter final score: 21
Pushups: 42

Test 2:
Enter final score: 0
Pushups: 0

Test 3:
Enter final score: 14
Pushups: 21

Retesting

All of the above test plus:

Test 4:
Enter final score: 42
Pushups: 147
Check floating-point values to 3 significant digits.

Test 1:

Enter x-coordinate of start of first line: 1.1
Enter y-coordinate of start of first line: 0.5
Enter x-coordinate of end of first line: 3.4
Enter y-coordinate of end of first line: -1.5
Enter x-coordinate of start of second line: 1.7
Enter y-coordinate of start of second line: -0.3
Enter x-coordinate of end of second line: 2.7
Enter y-coordinate of end of second line: 2.1
The line segments intersect at (1.7851065, -0.09574461).

Test 2:

Enter x-coordinate of start of first line: -1.2
Enter y-coordinate of start of first line: -1
Enter x-coordinate of end of first line: 1.3
Enter y-coordinate of end of first line: -1.3
Enter x-coordinate of start of second line: 0.5
Enter y-coordinate of start of second line: 0.6
Enter x-coordinate of end of second line: 2.1
Enter y-coordinate of end of second line: 1.4
The line segments do not intersect.

Retest:

Do the above tests, plus:

Test 3:

Enter x-coordinate of start of first line: -1
Enter y-coordinate of start of first line: -1
Enter x-coordinate of end of first line: 1
Enter y-coordinate of end of first line: 1
Enter x-coordinate of start of second line: 0
Enter y-coordinate of start of second line: 0
Enter x-coordinate of end of second line: 3.7
Enter y-coordinate of end of second line: -4.8
The line segments intersect at (0.0, 0.0).
4 Beginning — Matrix Multiplication

I/O elements *must* be labeled as shown.

**Test 1:**

- Enter a: 3
- Enter b: 5
- Enter c: -2
- Enter d: 6
- Enter e: -1
- Enter f: 7
- Enter g: 8
- Enter h: 4
- i = 37
- j = 41
- k = 50
- l = 10

**Test 2:**

- Enter a: 9
- Enter b: 8
- Enter c: 7
- Enter d: 6
- Enter e: 1
- Enter f: 0
- Enter g: 0
- Enter h: 1
- i = 9
- j = 8
- k = 7
- l = 6

**Retest:** Above tests plus:

**Test 3:**

- Enter a: 8
- Enter b: -2
- Enter c: -4
- Enter d: 6
- Enter e: -1
- Enter f: -7
- Enter g: 3
- Enter h: 2
- i = -14
- j = -60
- k = 22
- l = 40
5 Beginning - Patterns
Test Cases

Grading: the numbers must be the integers from 1 to 5

Test 1:
   Enter restricted value: 3
   Output: 1 3 2 5 4 - grading: any sequence is okay as long as 1 is not next to 2

Test 2:
   Enter restricted value: 10
   Output: 1 2 3 4 5 – grading: any sequence is okay

Test 3:
   Enter restricted value: 7
   Output: 1 3 5 4 2 – grading: 3 must not be next to 4 & 5 must not be next to 2

Retest:

All of the above tests plus

Test 4:
   Enter restricted value: 9
   Output: 5 2 3 4 1 – grading: any sequence as long as 4 is not next to 5
6 Beginning — Making Change
Test Cases

Test 1:

Enter value: 512
2000: 0
1000: 0
500: 1
100: 0
25: 0
10: 1
5: 0
1: 2

Test 2:

Enter value: 496
2000: 0
1000: 0
500: 0
100: 4
25: 3
10: 2
5: 0
1: 1

Test 3:

Enter value: 0
2000: 0
1000: 0
500: 0
100: 0
25: 0
10: 0
5: 0
1: 0

Retest:

Use the above tests, plus the following:

Test 4:

Enter value: 8128
2000: 4
1000: 0
500: 0
100: 1
25: 1
10: 0
5: 0
1: 3
1 Advanced - Sorting

Write a program that prompts for and reads in three pairs, each consisting of an upper-case letter and an integer, and then prints out the pairs in the following order:
- All pairs containing the same letter must be on the same line.
- On each line, the integers within the pairs must be in increasing order.
- The lines must be ordered alphabetically.

Example:

Enter first pair: A13
Enter second pair: B5
Enter third pair: A6

Output is A6 A13
Output is B5
2 Advanced - Willie’s Pushups

At KSU football games, Willie does pushups after each Wildcat score. After the first Wildcat touchdown (and point after), Willie does 7 pushups. After the second touchdown, the score is now 14 and Willie does 14 pushups.

Write a program that calculates the maximum number how many of pushups Willie may have to do during the game. Assume that only 3 point field goals and 7 point touchdowns (including the point after) occur. Prompt for the final score and print out the maximum number of pushups that Willie may have to do.

Example 1:
   Enter final score: 7
   Pushups: 7

Example 2:
   Enter final score: 13
   Pushups: 30

Example 3:
   Enter final score: 37
   Pushups: 242
3 Advanced — Convex Polygons

A *convex polygon* is a polygon whose interior angles are all strictly less than 180 degrees. For example, polygon a below is convex, but polygon b is not.

A convex polygon is a polygon whose interior angles are all strictly less than 180 degrees. For example, polygon a below is convex, but polygon b is not.

![Convex Polygons](image)

Write a program that takes a collection of real-valued coordinates of points and determines whether they can form a convex polygon. You may assume that there are at least 3 and at most 15 points. You may also assume that no two points have either the same x-coordinate or the same y-coordinate.

**Hint:** The perimeter of any polygon can be divided into two paths, each connecting the leftmost point with the rightmost point. In polygon c above, these two paths are a, b, c, d, e and a, h, g, f, e. In a convex polygon, the x-coordinates of the points in each of these paths must strictly increase in going from the leftmost point to the rightmost point. Let $D$ be the line segment connecting the leftmost point with the rightmost point (shown as the dashed line $ae$ in polygon c above). Then the intermediate points in one of the two paths must all be strictly above $D$, and the intermediate points in the other all must be strictly below $D$. Finally, as we go from left to right in the upper path, the slopes of the edges must strictly decrease, and as we go from left to right in the lower path, the slopes of the edges must strictly increase. If all of these conditions can be verified, the points form a convex polygon.

**Example 1:**

Enter the number of points: 3
Enter x0: 0
Enter y0: 0
Enter x1: 0.5
Enter y1: 2
Enter x2: -1
Enter y2: -0.7
These points form a convex polygon.

**Example 2:**

Enter the number of points: 4
Enter x0: 0
Enter y0: 0
Enter x1: -2.5
Enter y1: 1
Enter x2: 1
Enter y2: -2.5
Enter x3: 2
Enter y3: 2
These points do NOT form a convex polygon.
The product of an $n \times n$ matrix and an $n \times 1$ vector is defined to be

\[
\begin{pmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
    b_1 \\
    b_2 \\
    \vdots \\
    b_n
\end{pmatrix} =
\begin{pmatrix}
    c_1 \\
    c_2 \\
    \vdots \\
    c_n
\end{pmatrix}
\]

where each

\[
c_i = \sum_{j=1}^{n} a_{ij} b_j.
\]

A **Toeplitz matrix** is an $n \times n$ matrix such that for $1 < i \leq n$ and $1 < j \leq n$, $a_{ij} = a_{i-1,j-1}$; for example,

\[
\begin{pmatrix}
    1 & 4 & 6 \\
    2 & 1 & 4 \\
    5 & 2 & 1
\end{pmatrix}
\]

We can therefore describe an entire Toeplitz matrix by giving only the first column and the first row: $a_{nn}, a_{n,n-1}, \ldots, a_{11}, a_{12}, a_{1n}$. Write a program that takes as input a dimension $n$ followed by an $n \times n$ Toeplitz matrix described in the above fashion, followed by an $n \times 1$ vector, and produces the $n \times 1$ product.

Your input and output elements must be labeled as shown in the example below. You may assume that all elements are integers and that the dimension is at least 1 and at most 5.

**Example:**

Enter dimension: 3
Enter a31: 5
Enter a21: 2
Enter a11: 1
Enter a12: 4
Enter a13: 6
Enter b1: 1
Enter b2: 2
Enter b3: 3

\[
\begin{align*}
c_1 &= 27 \\
c_2 &= 16 \\
c_3 &= 12
\end{align*}
\]
5 Advanced - Patterns

We wish to construct a 3-by-3 array that contains the nine numbers, \( \text{Start} \) through \( \text{Start} + 8 \), such that no row or column has a sum equal to some given forbidden value \( \text{Bad} \). Write a program that prompts for the values \( \text{Start} \) and \( \text{Bad} \), then prints out an array that contains the 9 integers \( \text{Start} \) through \( \text{Start} + 8 \) arranged so that no row or column sums to \( \text{Bad} \). Note that there may be many patterns that satisfy the restrictions. Hint: pick a "standard" initial pattern and switch entries if it has wrong column or row sum.

Example 1:

Enter start value: 1
Enter restricted value: 12
Output: 7 8 9
        4 5 2
        3 1 6

Example 2:

Enter start value: 10
Enter restricted value: 36
Output: 10 11 12
        15 14 13
        16 17 18
The Stern-Brocot Tree is a construction consisting of infinitely many levels of nonnegative rational numbers, plus 1/0, which denotes infinity. Level 1 consists of the sequence (0/1, 1/0). For $i > 1$, level $i$ is constructed by inserting new elements between successive elements of level $i - 1$. Specifically, suppose $a/b$ and $c/d$ are successive elements in level $i - 1$. Then in level $i$, $(a + c)/(b + d)$ occurs between these elements. Thus, the first five levels are as follows:

\[
\begin{align*}
0/1 & \quad 1/0 \\
0/1 & \quad 1/1 & \quad 1/0 \\
0/1 & \quad 1/1 & \quad 2/1 & \quad 1/0 \\
0/1 & \quad 1/1 & \quad 2/1 & \quad 3/2 & \quad 1/1 & \quad 0/1 \\
0/1 & \quad 1/1 & \quad 2/1 & \quad 3/2 & \quad 1/2 & \quad 1/1 & \quad 0/1 \\
0/1 & \quad 1/1 & \quad 2/1 & \quad 3/2 & \quad 1/4 & \quad 3/5 & \quad 2/5 & \quad 3/4 & \quad 1/3 & \quad 2/3 & \quad 1/2 & \quad 1/1 & \quad 0/1 \\
\end{align*}
\]

Some remarkable facts about this construction are that every fraction constructed is already in lowest terms, the elements of each level are in increasing order, and every nonnegative rational number appears in some level.

Write a program that takes as input a level number and a position within that level, where the first element of the level is in position 1, and produces the fraction that occurs at the given position of the given level. **Your output must be a fraction, not a decimal approximation.** You may assume that the position is valid for that level.

Example 1:

Enter level number: 2
Enter position: 2
1/1

Example 2:

Enter level number: 4
Enter position: 6
3/2

Example 3:

Enter level number: 7
Enter position: 51
7/3
Write a program that prompts for and reads in three integers and then prints out the integers in numerical order, smallest first.

**Example:**

Enter first integer: 13
Enter second integer: 5
Enter third integer: 6

Output is 5 6 13
At KSU football games, Willie does pushups after each Wildcat score. After the first Wildcat touchdown (and point after), Willie does 7 pushups. After the second touchdown, the score is now 14 and Willie does 14 pushups.

Write a program that calculates how many pushups Willie does during the whole game. Assume that only 7 point touchdowns (including the point after) occur. Prompt for the final score and print out how many pushups Willie has done.

Example 1:
   Enter final score: 21
   Pushups: 42

Example 2:
   Enter final score: 7
   Pushups: 7
Write a program that takes as input the \(x\) and \(y\) coordinates of the endpoints of two line segments, and reports whether the two segments intersect. If they do intersect, it must print the intersection point. You should be able to handle floating-point numbers as input. You may assume that the two lines have different slope, and that neither is vertical.

Example 1:

Enter \(x\)-coordinate of start of first line: 0
Enter \(y\)-coordinate of start of first line: 0
Enter \(x\)-coordinate of end of first line: 1.5
Enter \(y\)-coordinate of end of first line: 1.5
Enter \(x\)-coordinate of start of second line: 1
Enter \(y\)-coordinate of start of second line: 0
Enter \(x\)-coordinate of end of second line: 0
Enter \(y\)-coordinate of end of second line: 1
The line segments intersect at (0.5, 0.5).

Example 2:

Enter \(x\)-coordinate of start of first line: 1
Enter \(y\)-coordinate of start of first line: 0
Enter \(x\)-coordinate of end of first line: 2
Enter \(y\)-coordinate of end of first line: 1
Enter \(x\)-coordinate of start of second line: 1.2
Enter \(y\)-coordinate of start of second line: 0.3
Enter \(x\)-coordinate of end of second line: 0
Enter \(y\)-coordinate of end of second line: 1
The line segments do not intersect.
4 Beginning — Matrix Multiplication

The product of two $2 \times 2$ matrices is defined as follows:

$$
\begin{pmatrix}
a & b \\
c & d
\end{pmatrix}
\begin{pmatrix}
e & f \\
g & h
\end{pmatrix} =
\begin{pmatrix}
i & j \\
k & l
\end{pmatrix}
$$

where

- $i = ae + bg$;
- $j = af + bh$;
- $k = ce + dg$; and
- $l = cf + dh$.

Write a program that takes as input two $2 \times 2$ arrays of integers and produces the product of these two arrays as output. The elements of the input and output arrays must be labeled as shown in the example below.

Example:

```
Enter a: 1
Enter b: 2
Enter c: 3
Enter d: 4
Enter e: 5
Enter f: 6
Enter g: 7
Enter h: 8
i = 19
j = 22
k = 43
l = 50
```
Write a program that arranges the integers 1 through 5 so that no two adjacent integers sum to a given restricted value. The program should first prompt for this restricted value and then print out the arrangement on a single line. Note that there may be many arrangements that satisfy the restrictions. For example, if 3 is the restricted value, then the arrangement 1 2 3 4 5 is not a valid output because 1 and 2 sum to 3; however, both of the arrangements 1 3 2 5 4 and 1 3 5 2 4 are valid outputs.

Example 1:
   Enter restricted value: 3
   Output: 1 3 2 5 4

Example 2:
   Enter restricted value: 10
   Output: 1 2 3 4 5
Write a program that takes a nonnegative integer monetary value as input and produces as output the number of each denomination of coin that is needed to achieve that value. The values of the coin denominations to be used are 1, 5, 10, 25, 50, 100, 500, 1000, and 2000. You must give preference to the higher denominations, so that a minimum number of coins is used; thus, even though there are several ways to achieve a value of 24, we must use as many 10s as we can (2), then as many 5s as we can (0), then as many 1s as we can (4).

Example 1:

Enter value: 24
2000: 0
1000: 0
500: 0
100: 0
25: 0
10: 2
5: 0
1: 4

Example 2:

Enter value: 2199
2000: 1
1000: 0
500: 0
100: 1
25: 3
10: 2
5: 0
1: 4
// Advanced - Sorting
// sortadvanced.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    char tempA;
    int tempB;
    cout << "Enter first char and number: ";
    char char1;
    int int1;
    cin >> char1; cin >> int1;
    cout << "Enter second char and number: ";
    char char2;
    int int2;
    cin >> char2; cin >> int2;
    cout << "Enter third char and number: ";
    char char3;
    int int3;
    cin >> char3; cin >> int3;

    if (char1 > char2)
    {
        tempA = char1;
        tempB = int1;
        char1 = char2;
        int1 = int2;
        char2 = tempA;
        int2 = tempB;
    }

    if (char2 > char3)
    {
        tempA = char2;
        tempB = int2;
        char2 = char3;
        int2 = int3;
        char3 = tempA;
        int3 = tempB;
    }

    if (char1 > char2)
    {
        tempA = char1;
        tempB = int1;
        char1 = char2;
        int1 = int2;
        char2 = tempA;
        int2 = tempB;
    }

    if (char1 == char2 && int1 > int2)
    {
        tempB = int1;
        int1 = int2;
        int2 = tempB;
    }
    else
    {
        cout << "Output is " << char1 << int1;
    }

    if (char2 == char3 && int2 > int3)
    {
        tempB = int2;
        int2 = int3;
        int3 = tempB;
    }
    else
    {
        cout << "Output is " << char2 << int2;
    }

    if (char1 == char2 && int1 > int2)
    {
        tempB = int1;
        int1 = int2;
        int2 = tempB;
    }
    else
    {
        cout << "Output is " << char1 << int1;
    }

    if (char1 == char2)
    {
        cout << "";
    }
    else
    {
        cout << "Output is " << char2 << int2;
    }

    if (char2 == char3)
    {
        cout << "";
    }
    else
    {
        cout << "Output is " << char3 << int3;
    }

    return 0;
}
// 2 Advanced --- Willie's Pushups

// willieadvanced.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    int score;
    int partscore;
    partscore = 0;
    int i;
    int pushups;
    pushups = 0;
    cout<<"enter score";
    cin>>score;
    int numberofsevens;
    numberofsevens = score % 3;

    if (numberofsevens == 1) pushups 7;
    if (numberofsevens == 2) pushups 21;

    for(i=3+7*numberofsevens;i<score+1;i=i+3)
    {
        pushups = pushups + i;
        partscore = i;
    }
    cout << "the number of pushups is " << pushups;

    return 0;
}
```java
package Advanced;

import java.io.*;

public class ConvexCheck {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter the number of points: ");
        int n = Integer.parseInt(in.readLine());
        Point[] points = new Point[n];
        for (int i = 0; i < n; i++) {
            points[i] = new Point();
            System.out.print("Enter x");
            points[i].x = Float.parseFloat(in.readLine());
            System.out.print("Enter y");
            points[i].y = Float.parseFloat(in.readLine());
        }
        sort(points);
        float dSlope = slope(points[0], points[n-1]);
        Point[] over = new Point[n];
        Point[] under = new Point[n];
        over[0] = points[0];
        under[0] = points[0];
        int numOver = 1;
        int numUnder = 1;
        for (int i = 1; i < n - 1; i++) {
            float s = slope(points[0], points[i]);
            if (s > dSlope) {
                over[numOver++] = points[i];
            } else if (s < dSlope) {
                under[numUnder++] = points[i];
            } else {
                System.out.println("These points do NOT form a convex polygon.");
                return;
            }
            over[numOver++] = points[n-1];
            under[numUnder++] = points[n-1];
            for (int i = 1; i < numOver - 1; i++) {
                if (slope(over[i-1], over[i]) < slope(over[i], over[i+1])) {
                    System.out.println("These points do NOT form a convex polygon.");
                    return;
                }
            }
            for (int i = 1; i < numUnder - 1; i++) {
                if (slope(under[i-1], under[i]) > slope(under[i], under[i+1])) {
                    System.out.println("These points do NOT form a convex polygon.");
                    return;
                }
            }
            System.out.println("These points form a convex polygon.");
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }

    public static void sort(Point[] p) {
        for (int i = 1; i < p.length; i++) {
            int j = i - 1;
            while (j > 0) && (p[j].x > p[j-1].x) {
                int t = j;
                p[j] = p[j-1];
                j--;
            }
            p[t] = p[j];
        }
    }

    public static float slope(Point a, Point b) {
        return (b.y - a.y)/(b.x - a.x);
    }
```
import java.io.*;

public class MatMultA {

    static BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));

    public static void main(String args[]) throws Exception {
        System.out.print("Enter dimension:");
        int n = Integer.parseInt(in.readLine());
        int a[] = new int[2*n-1];
        int b[] = new int[n];
        for (int i = 0; i < n; i++)
            a[i] = get("a" + (n - i) + "1");
        for (int i = n; i < 2*n - 1; i++)
            a[i] = get("a" + (i - n + 2));
        for (int i = 0; i < n; i++)
            b[i] = get("b" + (i + 1));
        for (int i = 0; i < n; i++) {
            int sum = 0;
            for (int j = 0; j < n; j++)
                sum = sum + a[n-i-1+j]*b[j];
            System.out.println("c" + (i+1) + "=" + sum);
        }
    }

    // Inputs the value described by s.
    static int get(String s) throws Exception {
        System.out.print("Enter " + s + ":");
        return Integer.parseInt(in.readLine());
    }
}
// 5 Advanced --- Patterns
// patternadvanced.cpp : Defines the entry point for the console application.

#include "sldatx.h"
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, TCHAR argv[])
{
    int intarray[3][3];
    int row[3], col[3];
    int sum;
cout << "Enter the starting value: ";
    int start;
    int value;
cin >> start;
    value = start;
cout << "Enter the restricted column/row/diagonal sum: ";
cin >> sum;
    int i, j;
    int temp;
    for(i=0; i<3; i++)
    {
        row[i] = 0;
        for(j=0; j<3; j++)
        {
            row[i] = row[i] + value;
            intarray[i][j] = value++;
        }
    }
    for(j=0; j<3; j++)
    {
        col[j] = 0;
        for(i=0; i<3; i++)
        {
            col[j] = col[j] + intarray[i][j];
        }
    }
    if(row[0] == sum) { temp = intarray[0][2];
        intarray[0][2] = intarray[1][2];
        intarray[1][2] = temp; }
    if(row[1] == sum) { temp = intarray[1][1];
        intarray[1][1] = intarray[2][2];
        intarray[2][2] = temp; }
    if(row[2] == sum) { temp = intarray[2][2];
        intarray[2][2] = intarray[1][1];
        intarray[1][1] = temp; }
    if(col[0] == sum) { temp = intarray[2][0];
        intarray[2][0] = intarray[1][1];
        intarray[1][1] = temp; }
    if(col[1] == sum) { temp = intarray[1][1];
        intarray[1][1] = intarray[2][2];
        intarray[2][2] = temp; }
    if(col[2] == sum) { temp = intarray[0][2];
        intarray[0][2] = intarray[0][1];
        intarray[0][1] = temp; }
    for(i=0; i<3; i++)
    {
        cout << end1;
        for(j=0; j<3; j++)
        {
            cout << intarray[i][j] << " ";
        }
    }
    return 0;
}
import java.io.*;

public class SternBrocot {

    public static void main(String args[]) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter level number:");
        int lev = Integer.parseInt(in.readLine());
        int[] oldnum = new int[2];
        int[] oldden = new int[2];
        oldnum[0] = 0;
        oldnum[1] = 1;
        oldden[0] = 1;
        oldden[1] = 0;
        for (int i = 1; i < lev; i++) {
            int num[] = new int[2*oldnum.length - 1];
            int den[] = new int[num.length];
            for (int j = 0; j < oldnum.length - 1; j++) {
                num[2*j] = oldnum[j];
                num[2*j+1] = oldnum[j] + oldnum[j+1];
                den[2*j] = oldden[j];
                den[2*j+1] = oldden[j] + oldden[j+1];
            }
            num[num.length-1] = 1;
            den[den.length-1] = 0;
            oldnum = num;
            oldden = den;
        }
        System.out.print("Enter position:");
        int pos = Integer.parseInt(in.readLine());
        System.out.println(oldnum[pos-1] + "/" + oldden[pos-1]);
    }
}
// 1 Beginning --- Sorting

// sortbeginner.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[]) {
    int temp;
    cout<<"enter first number: ";
    int int1;
    cin>>int1;
    cout<<"enter second number: ";
    int int2;
    cin>>int2;
    cout<<"enter third number: ";
    int int3;
    cin>>int3;

    if (int1 > int2)
    {
        temp = int1;
        int1 = int2;
        int2 = temp;
    }
    if (int2 > int3)
    {
        temp = int2;
        int2 = int3;
        int3 = temp;
    }
    if (int1 > int2)
    {
        temp = int1;
        int1 = int2;
        int2 = temp;
    }

    cout << "\noutput is"<<int1 <<" " <<int2;
    cout << " " << int3 << endl;

    return 0;
}
// 2 Beginning - Willie's Pushups

// williebeginner.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, TCHAR* argv[])
{
    int score;
    int i;
    int pushups;
    pushups = 0;
    cout<<"enter score";
    cin>>score;

    for(i=7;i<score+1;i=i+7) {
        pushups = pushups + i;
    }
    cout << "the number of pushups is "<< pushups;

    return 0;
}
// 3 Beginning --- Line Segments

import java.io.*;

public class LineSegments {

    static BufferedReader in
        = new BufferedReader(new InputStreamReader(System.in));

    public static void main(String args[]) throws Exception {
        float startxl = get("start", "x", "first");
        float startyl = get("start", "y", "first");
        float endxl = get("end", "x", "first");
        float endyl = get("end", "y", "first");
        float startx2 = get("start", "x", "second");
        float starty2 = get("start", "y", "second");
        float endx2 = get("end", "x", "second");
        float endy2 = get("end", "y", "second");

        // Slope = rise / run
        float m1 = (endyl - startyl) / (endxl - startxl);
        float m2 = (endy2 - starty2) / (endx2 - startx2);

        // Find y-intercepts
        float b1 = startyl - m1 * startxl;
        float b2 = starty2 - m2 * startx2;

        // The lines cross when m1*x + b1 = m2*x + b2
        //
        float crossx = (b2 - b1) / (m1 - m2);
        float crossy = m1 * crossx + b1;

        // Does the intersection occur between the endpoints of both lines?
        if ((Math.abs(startxl - crossx) + Math.abs(crossx - endxl)) ==
            Math.abs(startxl - endxl)) &&
            (Math.abs(startx2 - crossx) + Math.abs(crossx - endx2)) ==
            Math.abs(startx2 - endx2))
            System.out.println("The line segments intersect at(" + crossx +
            "," + crossy + ").");
        else
            System.out.println("The line segments do not intersect.");
    }

    /**
     * Supply a prompt containing the given strings and input a number.
     */
    public static float get(String end, String coord, String line)
        throws Exception {
        System.out.print("Enter " + coord + "-coordinate of " +
            end + " of " + line + " line:");
        return Float.parseFloat(in.readLine());
    }
}
// 4 Beginning - Matrix Multiplication
///
// Note: a somewhat cleaner solution uses arrays.
import java.io.*;

public class MatMultB {

    public static void main(String args[]) throws Exception {
        int a = get("a");
        int b = get("b");
        int c = get("c");
        int d = get("d");
        int e = get("e");
        int f = get("f");
        int g = get("g");
        int h = get("h");
        put("i", a*e + b*g);
        put("j", a*f + b*h);
        put("k", c*e + d*g);
        put("l", c*f + d*h);
    }

    /* Inputs the value described by s. */
    public static int get(String s) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter " + s + ":");
        return Integer.parseInt(in.readLine());
    }

    /* Prints c with label s. */
    public static void put(String s, int c) {
        System.out.println(s + " = " + c);
    }
}
// 5 Beginning --- Patterns

// patternbeginner.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    int int1 = 1;
    int int2 = 2;
    int int3 = 3;
    int int4 = 4;
    int int5 = 5;
    int sum;
    int temp;

    cout << "enter forbidden sum: ";
    cin >> sum;
    if (int1 + int2 == sum) {
        temp = int2;
        int2 = int3;
        int3 = temp;
    }
    if (int2 + int3 == sum) {
        temp = int3;
        int3 = int4;
        int4 = temp;
    }
    if (int3 + int4 == sum) {
        temp = int4;
        int4 = int5;
        int5 = temp;
    }
    if (int4 + int5 == sum) {
        temp = int5;
        int5 = int1;
        int1 = temp;
    }
    cout << "pattern is " << int1 << int2 << int3 << int4 << int5;

    return 0;
}
// 6 Beginning - Making Change

import java.io.*;

public class CoinsB {

    public static void main(String args[]) throws Exception {
        int denom[] = new int[] {2000, 1000, 500, 100, 25, 10, 5, 1};
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter value: ");
        int v = Integer.parseInt(in.readLine());
        for (int i = 0; i < denom.length; i++) {
            int num = v/denom[i];
            System.out.println(denom[i] + "": " + num);
            v = v - num*denom[i];
        }
    }
}